Dynamic Frames in Java Dynamic Logic
Formalisation and Proofs

Peter H. Schmitt, Mattias Ulbrich, Benjamin Weiß

2010
Please note:
This Report has been published on the Internet under the following Creative Commons License:
http://creativecommons.org/licenses/by-nc-nd/3.0/de.
Dynamic Frames in Java Dynamic Logic
Formalisation and Proofs

Peter H. Schmitt, Mattias Ulbrich, and Benjamin Weiß
Karlsruhe Institute of Technology
Institute for Theoretical Computer Science
D-76128 Karlsruhe, Germany
{pschmitt,mulbrich,bweiss}@ira.uka.de

Abstract. This report is a companion to the paper Dynamic Frames in Java Dynamic Logic [2]. It contains complementary formal definitions and proofs.

1 Formalisation

1.1 Syntax

Definition 1 (Signatures). A signature \( \Sigma \) is a tuple

\[
\Sigma = (\mathcal{T}, \sqsubseteq, V, PV, F, P, \alpha, Prg)
\]

where \( \mathcal{T} \) is a finite set of types; where \( \sqsubseteq \) is a partial order on \( \mathcal{T} \) called the subtype relation; where \( V \) is a set of (logical) variables; where \( PV \) is a set of program variables; where \( F \) is a set of function symbols; where \( P \) is a set of predicate symbols; where \( \alpha \) is a static typing function such that \( \alpha(v) \in \mathcal{T} \) for all \( v \in V \cup PV \), \( \alpha(f) \in \mathcal{T}^* \times \mathcal{T} \) for all \( f \in F \), and \( \alpha(p) \in \mathcal{T}^* \) for all \( p \in P \); and

where \( Prg \) is some Java program, i.e., a set of Java classes and interfaces.

We use the notation \( v : A \) for \( \alpha(v) = A \), the notation \( f : A_1, \ldots, A_n \rightarrow A \) for \( \alpha(f) = ((A_1, \ldots, A_n), A) \), and the notation \( p : A_1, \ldots, A_n \) for \( \alpha(p) = (A_1, \ldots, A_n) \).

We require that the following types, program variables, function and predicate symbols are present in every signature:

- Any, Boolean, Int, Null, LocSet, Field, Heap \( \in \mathcal{T} \)
- all reference types of \( Prg \) also appear as types in \( \mathcal{T} \); in particular, Object \( \in \mathcal{T} \)
- all local variables \( a \) of \( Prg \) with Java type \( T \) also appear as program variables \( a : A \in PV \), where \( A = T \) if \( T \) is a reference type, \( A = \text{Boolean} \) if \( T = \text{boolean} \), and \( A = \text{Int} \) if \( T = \text{int} \) (in this paper we do not consider other primitive types, and we ignore integer overflows)
- \( \text{heap} : \text{Heap} \in PV \)
- \( \text{cast}_A : \text{Any} \rightarrow A \in F \) (for every type \( A \in \mathcal{T} \))
- TRUE, FALSE : Boolean \( \in F \)
- \( \text{select}_A : \text{Heap}, \text{Object}, \text{Field} \rightarrow A \in F \) (for every type \( A \in \mathcal{T} \))
We also require that Boolean, Int, Object, LocSet ⊑ Any; that for all C ∈ \(\mathcal{T}\) with
\(C \subseteq Object\) we have Null ⊑ C; that for all types \(A, A'\) of Prg we have \(A' \subseteq A\)
if and only if \(A'\) is a subtype of \(A\) in Prg; that the types explicitly mentioned
in this definition are otherwise unrelated to each other wrt. \(\subseteq\); and that
the types Boolean, Int, Null, LocSet, Field and Heap do not have subtypes except
each other. Finally, we demand that \(\mathcal{V}, \mathcal{PV}, \mathcal{F}\) and \(\mathcal{P}\) each contain an infinite
number of symbols of every typing.

For illustration, the type hierarchy is visualised in Fig. 1. In the following, we
assume a fixed signature \(\Sigma = (\mathcal{T}, \subseteq, \mathcal{V}, \mathcal{PV}, \mathcal{F}, \mathcal{P}, \alpha, \text{Prg})\).

\begin{center}
\begin{tikzpicture}
  \node (Any) {Any};
  \node (Field) [below left of=Any] {Field};
  \node (Heap) [below right of=Any] {Heap};
  \node (Boolean) [below left of=Field] {Boolean};
  \node (Int) [below left of=Boolean] {Int};
  \node (Object) [below left of=Int] {Object};
  \node (LocSet) [below right of=Field] {LocSet};
  \node (Null) [below right of=LocSet] {Null};
  \node (...1) [above right of=Field] {...};
  \node (...2) [above right of=Heap] {...};

  \draw[->] (Any) -- (Field);
  \draw[->] (Any) -- (Heap);
  \draw[->] (Any) -- (Boolean);
  \draw[->] (Any) -- (Int);
  \draw[->] (Any) -- (Object);
  \draw[->] (Any) -- (LocSet);
  \draw[->] (Any) -- (Null);
  \draw[->] (Field) -- (...1);
  \draw[->] (Heap) -- (...2);

\end{tikzpicture}
\end{center}

\textbf{Fig. 1. Type hierarchy}

\begin{definition}[Syntax] \(\text{Trm}_\Sigma^A\) of terms of type \(A\), \(\text{Fma}_\Sigma\) of formulas
and \(\text{Upd}_\Sigma\) of updates are defined by the following grammar:

\[
\text{Trm}_\Sigma^A ::= x \mid a \mid f(\text{Trm}_\Sigma^{B_1}, \ldots, \text{Trm}_\Sigma^{B_n}) \mid \text{if}(\text{Fma}_\Sigma)\text{then}(\text{Trm}_\Sigma^A)\text{else}(\text{Trm}_\Sigma^A) \mid \{\text{Upd}_\Sigma\} \text{Trm}_\Sigma^A
\]

\[
\text{Fma}_\Sigma ::= \text{true} \mid \text{false} \mid p(\text{Trm}_\Sigma^{B_1}, \ldots, \text{Trm}_\Sigma^{B_n}) \mid \neg\text{Fma}_\Sigma \mid \text{Fma}_\Sigma \land \text{Fma}_\Sigma \mid \text{Fma}_\Sigma \lor \text{Fma}_\Sigma \mid \alpha(\text{Fma}_\Sigma) \mid \text{Fma}_\Sigma \rightarrow \text{Fma}_\Sigma \mid \text{Fma}_\Sigma \leftrightarrow \text{Fma}_\Sigma
\]
\end{definition}
\[ \forall A : \text{Fma}_\Sigma \mid \exists A : \text{Fma}_\Sigma \mid [p]\text{Fma}_\Sigma \mid \langle p \rangle \text{Fma}_\Sigma \mid \{\text{Upd}_\Sigma\}\text{Fma}_\Sigma \]

\[
\text{Upd}_\Sigma := \text{a} := \text{Trm}_\Sigma^A \mid \text{Upd}_\Sigma \parallel \text{Upd}_\Sigma \mid \{\text{Upd}_\Sigma\}\text{Upd}_\Sigma
\]

for any variable \( x : A \in \mathcal{V} \), any program variable \( \text{a} : A \in \mathcal{P} \mathcal{V} \), any function symbol \( f : B_1, \ldots, B_n \to A \in \mathcal{F} \) and any predicate symbol \( p : B_1, \ldots, B_n \) where \( B_1' \subseteq B_1, \ldots, B_n' \subseteq B_n \), any executable Java fragment \( p \), and any type \( A' \in \mathcal{T} \) with \( A' \sqsubseteq A \).

A sequent is a syntactical construct \( \Gamma \Rightarrow \Delta \), where \( \Gamma, \Delta \in 2^{\text{Fma}_\Sigma} \) are finite sets of formulas.

We require that every Kripke structure satisfies the following:

- \( \mathcal{D}^\text{Boolean} = \{tt, ff\}, \mathcal{D}^\text{Int} = \mathbb{Z}, \mathcal{D}^\text{Null} = \{I(\text{null})\}, \mathcal{D}^\text{LocSet} = 2^{\mathcal{D}^\text{Object} \times \mathcal{D}^\text{Field}}, \mathcal{D}^\text{Heap} = \mathcal{D}^\text{Object} \times \mathcal{D}^\text{Field} \rightarrow \mathcal{D}^\text{Any} \)
- \( \delta(d) \neq T \) for all \( d \in \mathcal{D} \), if \( T \in \mathcal{T} \) represents an interface or an abstract class
- \( \{d \in \mathcal{D} | \delta(d) = T\} \) is infinite for all \( T \sqsubseteq \text{Object}, T \neq \text{Null} \) not representing an interface or an abstract class
- \( I(\text{cast}_A)(d) = d \) for all \( d \in \mathcal{D}^A \)
- \( I(\text{TRUE}) = tt, I(\text{FALSE}) = ff \)
- \( I(\text{select}_A)(h, o, f) = I(\text{cast}_A)(h(o, f)) \) for all \( h \in \mathcal{D}^\text{Heap}, o \in \mathcal{D}^\text{Object}, f \in \mathcal{D}^\text{Field} \)

1.2 Semantics

**Definition 3 (Kripke structures).** A Kripke structure \( \mathcal{K} \) for a signature \( \Sigma \) is a tuple

\[ \mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho) \]

where \( \mathcal{D} \) is a set of semantical values called the domain; where \( \delta \) is a dynamic typing function \( \delta : \mathcal{D} \rightarrow \mathcal{T} \); where (using the definition \( \mathcal{D}^A = \{d \in \mathcal{D} | \delta(d) \subseteq A\}\)) \( I \) is an interpretation function that maps every function symbol \( f : A_1, \ldots, A_n \rightarrow A \in \mathcal{F} \) to a function \( I(f) : \mathcal{D}^{A_1} \times \cdots \times \mathcal{D}^{A_n} \rightarrow \mathcal{D}^A \) and every predicate symbol \( p : A_1, \ldots, A_n \in \mathcal{P} \) to a relation \( I(p) \subseteq \mathcal{D}^{A_1} \times \cdots \times \mathcal{D}^{A_n} \); where \( \mathcal{S} \) is the set of all states, which are functions \( s \in \mathcal{S} \) mapping every program variable \( \text{a} : A \in \mathcal{P} \mathcal{V} \) to a value \( s(\text{a}) \in \mathcal{D}^A \); and where \( \rho \) is a function associating with every executable Java fragment \( p \) in the context of \( \text{Prg} \) a transition relation \( \rho(p) \subseteq \mathcal{S}^2 \) such that \( (s_1, s_2) \in \rho(p) \) iff \( p \), when started in \( s_1 \), terminates normally in \( s_2 \) (according to the Java semantics [1]). We consider Java programs to be deterministic, so for all program fragments \( p \) and all \( s_1 \in \mathcal{S} \), there is at most one \( s_2 \) such that \( (s_1, s_2) \in \rho(p) \).

We require that every Kripke structure satisfies the following:

- \( \mathcal{D}^\text{Boolean} = \{tt, ff\}, \mathcal{D}^\text{Int} = \mathbb{Z}, \mathcal{D}^\text{Null} = \{I(\text{null})\}, \mathcal{D}^\text{LocSet} = 2^{\mathcal{D}^\text{Object} \times \mathcal{D}^\text{Field}}, \mathcal{D}^\text{Heap} = \mathcal{D}^\text{Object} \times \mathcal{D}^\text{Field} \rightarrow \mathcal{D}^\text{Any} \)
- \( \delta(d) \neq T \) for all \( d \in \mathcal{D} \), if \( T \in \mathcal{T} \) represents an interface or an abstract class
- \( \{d \in \mathcal{D} | \delta(d) = T\} \) is infinite for all \( T \sqsubseteq \text{Object}, T \neq \text{Null} \) not representing an interface or an abstract class
- \( I(\text{cast}_A)(d) = d \) for all \( d \in \mathcal{D}^A \)
- \( I(\text{TRUE}) = tt, I(\text{FALSE}) = ff \)
- \( I(\text{select}_A)(h, o, f) = I(\text{cast}_A)(h(o, f)) \) for all \( h \in \mathcal{D}^\text{Heap}, o \in \mathcal{D}^\text{Object}, f \in \mathcal{D}^\text{Field} \)
Given a Kripke structure 

- \( I(\text{store})(h,o,f,d)(o',f') = \begin{cases} 
  d & \text{if } o = o' \text{ and } f = f' \\
  h(o',f') & \text{otherwise} 
\end{cases} \)

for all \( h \in D_{\text{Heap}}, o, o' \in D_{\text{Object}}, f, f' \in D_{\text{Field}}, d \in D_{\text{Any}} \)

- \( I(\text{anon})(h,s,h')(o,f) = \begin{cases} 
  h'(o,f) & \text{if } ((o,f) \in s \text{ and } f \neq I(\text{created})) \\
  h(o,f) & \text{otherwise} 
\end{cases} \)

for all \( h, h' \in D_{\text{Heap}}, s \in D_{\text{LocSet}}, o \in D_{\text{Object}}, f \in D_{\text{Field}} \)

- let \( \text{UniqueFunctions} \subseteq \mathcal{F} \) be the set consisting of the constant symbols representing Java fields, or \( \text{arr} \) and of \( \text{created} \); then we require that for all \( f, g \in \text{UniqueFunctions} \) the function \( I(f) \) is injective, and that the ranges of the functions \( I(f) \) and \( I(g) \) are disjoint.

- \( I(\text{allLocs}) = D_{\text{Object}} \times D_{\text{Field}}, I(\text{allFields})(o) = \{(o,f) | f \in D_{\text{Field}}\}, I(\text{freshLocs})(h) = \{(o,f) \in I(\text{allLocs}) | o \neq I(\text{null}), h(o,I(\text{created})) = ff\} \)

- \( I(\emptyset) = \emptyset, I(\text{singleton})(o,f) = \{(o,f)\}, I(\cup) = \cup, I(\cap) = \cap, I(\setminus) = \setminus \)

- \( I(\text{exactInstance}_A) = \{d \in D | \delta(d) = A\} \)

- \( I(\text{wellFormed}) = \{h \in D_{\text{Heap}} | \text{for all } o \in D_{\text{Object}}, f \in D_{\text{Field}}; \) \( \text{if } h(o,f) \in D_{\text{Object}}, \text{ then } h(o,f) = I(\text{null}) \) \( \text{or } h(h(o,f), I(\text{created})) = tt\} \)

\[
\begin{align*}
I(\hat{=} & ) = \{(d,d) \in D^2\} \\
I(\hat{\in} & ) = \{(o,f,s) \in D_{\text{Object}} \times D_{\text{Field}} \times D_{\text{LocSet}} | (o,f) \in s\}, I(\hat{\cap}) = \{(s_1,s_2) \in (D_{\text{LocSet}})^2 | s_1 \subseteq s_2\}, I(\text{disjoint}) = \{(s_1,s_2) \in (D_{\text{LocSet}})^2 | s_1 \cap s_2 = \emptyset\}
\end{align*}
\]

**Definition (Semantics).** Given a Kripke structure \( K = (D,\delta,I,S,\rho) \), a state \( s \in S \) and a variable assignment \( \beta : V \to D \) (where for every \( x : A \in \mathcal{V} \) we have \( \beta(x) \in D_A \)), we evaluate terms \( t \in \text{Trm}^A \) to a value \( \text{val}_{K,s,\beta}(t) \in D_A \), formulas \( \varphi \in \text{Fma}^\Sigma \) to a truth value \( \text{val}_{K,s,\beta}(\varphi) \in \{tt,ff\} \), and updates \( u \in \text{Upd}^\Sigma \) to a state transformer \( \text{val}_{K,s,\beta}(u) : S \to S \) as defined below.

\[
\begin{align*}
\text{val}_{K,s,\beta}(x) & = \beta(x) \\
\text{val}_{K,s,\beta}(a) & = s(a) \\
\text{val}_{K,s,\beta}(f(t_1,\ldots,t_n)) & = I(f)(\text{val}_{K,s,\beta}(t_1),\ldots,\text{val}_{K,s,\beta}(t_n)) \\
\text{val}_{K,s,\beta}(\text{if}(\varphi)\text{then}(t_1)\text{else}(t_2)) & = \begin{cases} 
\text{val}_{K,s,\beta}(t_1) & \text{if } \text{val}_{K,s,\beta}(\varphi) = tt \\
\text{val}_{K,s,\beta}(t_2) & \text{otherwise} 
\end{cases} \\
\text{val}_{K,s,\beta}((u)t) & = \text{val}_{K,s',\beta}(t), \text{ where } s' = \text{val}_{K,s,\beta}(u)(s) \\
\text{val}_{K,s,\beta}(\text{true}) & = tt \\
\text{val}_{K,s,\beta}(\text{false}) & = ff \\
\text{val}_{K,s,\beta}(p(t_1,\ldots,t_n)) & = tt \text{ iff } (\text{val}_{K,s,\beta}(t_1),\ldots,\text{val}_{K,s,\beta}(t_n)) \in I(p) \\
\text{val}_{K,s,\beta}(\neg\varphi) & = tt \text{ if } \text{val}_{K,s,\beta}(\varphi) = ff \\
\text{val}_{K,s,\beta}(\varphi_1 \land \varphi_2) & = tt \text{ iff } ff \notin \{\text{val}_{K,s,\beta}(\varphi_1), \text{val}_{K,s,\beta}(\varphi_2)\} \\
\text{val}_{K,s,\beta}(\varphi_1 \lor \varphi_2) & = tt \text{ iff } tt \in \{\text{val}_{K,s,\beta}(\varphi_1), \text{val}_{K,s,\beta}(\varphi_2)\} \\
\text{val}_{K,s,\beta}(\varphi_1 \rightarrow \varphi_2) & = \text{val}_{K,s,\beta}(\neg\varphi_1 \lor \varphi_2) \\
\text{val}_{K,s,\beta}(\varphi_1 \leftrightarrow \varphi_2) & = \text{val}_{K,s,\beta}(\varphi_1 \rightarrow \varphi_2 \land \varphi_2 \rightarrow \varphi_1)
\end{align*}
\]
We sometimes write $K$ instead of $\text{val}_{K,s,\beta}(\varphi) = \text{tt}$. A formula $\varphi \in F_{\Sigma}$ is called logically valid, in symbols $\models \varphi$, iff $(K,s,\beta) \models \varphi$ for all Kripke structures $K$, all states $s \in S$, and all variable assignments $\beta$.

The semantics of a sequent $\Gamma \Rightarrow \Delta$ is the same as that of a formula $\bigwedge \Gamma \Rightarrow \bigvee \Delta$, where $\bigvee \{\varphi_1, \ldots, \varphi_n\} = \varphi_1 \lor \cdots \lor \varphi_n$, and $\bigwedge \{\varphi_1, \ldots, \varphi_n\} = \varphi_1 \land \cdots \land \varphi_n$.

1.3 Observations

The propositions below are used as assumptions in the proofs in Sect. 2. We do not prove them, but consider them obvious.

**Proposition 1 (Non-occurring program variables).** For all Kripke structures $K$, all states $s, s' \in S$, all variable assignments $\beta$, and all $t \in \text{Trm}_\Sigma \cup F_{\Sigma} \cup \text{Upd}_{\Sigma}$: if for all program variables $a \in \mathcal{P} \lor \mathcal{V}$ that syntactically occur in $t$ we have $s(a) = s'(a)$, then we also have $\text{val}_{K,s,\beta}(t) = \text{val}_{K,s',\beta}(t)$.

Note that a program variable $b$ not occurring in $t$ can play a role in evaluating $t$, namely if $t$ contains a program which calls a method that in turn manipulates $b$. Still, in a Java program a called method can never read the value of a local variable $b$ before assigning to $b$; thus, the initial value of $b$ as defined by $s$ or $s'$ does not matter. We consider $\text{heap} \in \mathcal{P} \lor \mathcal{V}$ to implicitly occur in field access expressions $o.f$, in array access expressions $a[i]$, and in method calls $o.m(\ldots)$.

**Proposition 2 (Non-occurring function and predicate symbols).** For all Kripke structures $K = (D, \delta, I, S, \rho)$ and $K' = (D, \delta', I', S, \rho)$ differing only in the interpretation functions $I$ vs. $I'$, all states $s \in S$, all variable assignments $\beta$, and all $t \in \text{Trm}_\Sigma \cup F_{\Sigma} \cup \text{Upd}_{\Sigma}$: if for all function and predicate symbols $f \in \mathcal{F} \lor \mathcal{P}$ that syntactically occur in $t$ we have $I(f) = I'(f)$, then we also have $\text{val}_{K,s,\beta}(t) = \text{val}_{K',s,\beta}(t)$.

**Proposition 3 (Overwritten program variables).** For all Kripke structures $K$, all states $s, s' \in S$, all variable assignments $\beta$, all updates $(a := t') \in \text{Upd}_{\Sigma}$,
where \( a \) does not occur in \( t' \), all \( t \in \text{Trm}_\Sigma \cup \text{Fma}_\Sigma \cup \text{Upd}_\Sigma \), all \( \varphi \in \text{Fma}_\Sigma \), and all program fragments \( p \): if for all program variables \( b \in \mathcal{PV} \setminus \{ a \} \) which occur in \( t \) or \( \varphi \) we have \( s(b) = s'(b) \), then we also have:

\[
\begin{align*}
\text{val}_{K,s,\beta}(\{a := t'\}t) &= \text{val}_{K,s',\beta}(\{a := t'\}t) \\
\text{val}_{K,s,\beta}(\{a = t'; p\}\varphi) &= \text{val}_{K,s',\beta}(\{a = t'; p\}\varphi) \\
\text{val}_{K,s,\beta}(\langle a = t'; p \rangle \varphi) &= \text{val}_{K,s',\beta}(\langle a = t'; p \rangle \varphi)
\end{align*}
\]

Prop. 3 holds because the initial value of the program variable \( a \) is overwritten by the preceding update or assignment, and thus cannot influence the evaluation of \( t \) or \( \varphi \), respectively.

**Proposition 4 (Method calls).** Let \( p \) be a method call statement \((\text{res} = \text{this.m}(p_1, \ldots, p_n);)\), let \( h_{\text{Pre}} : \text{Heap} \in \mathcal{PV} \), let \( \text{reachableState} \in \text{Fma}_\Sigma \) be as in Def. 3 of [2], let \( \text{reachableState}' \in \text{Fma}_\Sigma \) be as in Def. 4, and let \( \text{noDeallocs} \in \text{Fma}_\Sigma \) be as in Def. 7. Then the following holds:

\[
\models \text{reachableState} \rightarrow \{h_{\text{Pre}} := \text{heap}\}[p](\text{reachableState}' \land \text{noDeallocs})
\]

Prop. 4 is guaranteed by the semantics of Java.

**2 Proofs**

**2.1 Preparation**

**Lemma 1 (Relation between frame and anon).** Let \( \text{mod} \in \text{Trm}^{\mathcal{LocSet}}_\Sigma \), \( h_{\text{Pre}} : \text{Heap} \in \mathcal{PV} \), frame \( \in \text{Fma}_\Sigma \) be as in Def. 3 of [2], \( \text{noDeallocs} \in \text{Fma}_\Sigma \) be as in Def. 7, and let \( \text{frame}' \in \text{Fma}_\Sigma \) be the formula

\[
\text{heap} \triangleq \text{anon}(h_{\text{Pre}}, \{\text{heap} := h_{\text{Pre}}\}\text{mod}, \text{heap}).
\]

Then the following holds:

\[
\models (\text{frame} \land \text{noDeallocs}) \leftrightarrow \text{frame}'
\]

**Proof.** Let \( K \) be a Kripke structure, \( s \in S \) be a state, \( \beta \) be a variable assignment, \( h = s(\text{heap}) \), \( h' = \text{val}_{K,s,\beta}(\text{anon}(h_{\text{Pre}}, \{\text{heap} := h_{\text{Pre}}\}\text{mod}, \text{heap}), s^{\text{pre}} = \text{val}_{K,s,\beta}(\text{heap} := h_{\text{Pre}})(s), h^{\text{pre}} = s^{\text{pre}}(\text{heap}), m^{\text{pre}} = \text{val}_{K,s,\beta}(\text{mod}), f = I(\text{freshLocs})(h), \) and \( f^{\text{pre}} = I(\text{freshLocs})(h^{\text{pre}}) \). Note that \( h^{\text{pre}} = s(h_{\text{Pre}}) \).

By definition of \( I(\text{anon}) \), we know that the following holds for all \( o \in \mathcal{D}^{\text{Object}}, f \in \mathcal{D}^{\text{Field}}, \)

\[
h'(o,f) = \begin{cases} h(o,f) & \text{if } ((o,f) \in m^{\text{pre}} \text{ and } f \neq I(\text{created})) \\ h^{\text{pre}}(o,f) & \text{otherwise} \end{cases}
\]

(1)

We first show that \( (K,s,\beta) \models \text{frame} \land \text{noDeallocs} \) implies that \( (K,s,\beta) \models \text{frame}' \), and then the other way round.
1. Let \( o \in D^{Object} \), \( f \in D^{Field} \). Using the definitions of frame, noDeallocs and \( frame' \), we assume

\[
(o, f) \in m^{\pre} \cup f^{\pre} \quad \text{or} \quad h(o, f) = h^{\pre}(o, f) \tag{2}
\]

if \((o, f) \in fl\), then \((o, f) \in f^{\pre} \) \tag{3}

\[
h(I(null), I(created)) = h^{\pre}(I(null), I(created)) \tag{4}
\]

and aim to show

\[
h'(o, f) = h(o, f). \tag{5}
\]

From (2) we get that one of the following three cases must apply:

- \((o, f) \in m^{\pre}\). If \( f \neq I(created) \) or \((o, f) \in f^{\pre}\), then (5) immediately follows from (1). We thus assume

\[
f = I(created) \tag{6}
\]

\[
(o, f) \notin f^{\pre}. \tag{7}
\]

Now, (1) yields

\[
h'(o, f) = h^{\pre}(o, f). \tag{8}
\]

If \( o = I(null) \), then we get from (4) that \( h(o, f) = h^{\pre}(o, f) \), which together with (8) immediately yields (5). Thus we assume

\[
o \neq I(null). \tag{9}
\]

From (3) and (7) we get that

\[
(o, f) \notin fl.
\]

This, (9), and the definition of \( I(freshLocs) \) imply \( h(o, I(created)) = tt \). Analogously, (7) and (9) imply \( h^{\pre}(o, I(created)) = tt \). Together, we have \( h(o, I(created)) = h^{\pre}(o, I(created)) \), which because of (6) can be written as \( h(o, f) = h^{\pre}(o, f) \). We combine this with (8) to get (5).

- \((o, f) \in f^{\pre}\). Then (1) immediately yields (5).

- \( h(o, f) = h^{\pre}(o, f) \). If \((o, f) \in m^{\pre}\) or \((o, f) \in f^{\pre}\), then the proof proceeds as for the respective case above. Otherwise, (1) guarantees that \( h'(o, f) = h^{\pre}(o, f) \), and thus we have (5).

2. Let \( o \in D^{Object}, f \in D^{Field} \). We assume (5), and show first (2), then (3), and finally (4).

(a) If \((o, f) \in m^{\pre}\) or \((o, f) \in f^{\pre}\), then (2) holds trivially. Otherwise, (5) and (1) imply \( h(o, f) = h^{\pre}(o, f) \), which also implies (2).

(b) We prove (3) by contradiction: we assume that \((o, f) \in f \setminus f^{\pre}\). By definition of \( I(freshLocs) \), this means that \( o \neq I(null) \), that \( h(o, I(created)) = ff \), and that \( h^{\pre}(o, I(created)) = tt \). From (5) and (1) we get that \( h(o, I(created)) = h^{\pre}(o, I(created)) \). Together, we have \( ff = tt \).

(c) The definition of \( I(freshLocs) \) tells us that \((I(null), I(created)) \notin f^{\pre}\). Thus, (5) and (1) immediately guarantee (4). \( \square \)
2.2 Method Contracts

Theorem 1 (Soundness of useMethodContract). Let $\Gamma, \Delta \in 2^{F_{ma, o}}$, $u \in Upd_S$, $[\cdot] \in \{\cdot, \}, r \in PV$, $o \in Trm_S$, the method $m$, $p'_1, \ldots, p'_n \in Trm_S$, $\varphi \in Fma_S$, $A \in T$, $mct = (m, this, (p_1, \ldots, p_n), res, hPre, pre, post, mod, r)$, reachableState, reachableState' $\in Fma_S$, $v, w \in Upd_S$, and $h, r' \in F$ all be as in Def. 4 of [2]. If

$$\models \Gamma \vdash \{u\}\{w\}(pre \land \text{reachableState}), \Delta \tag{10}$$

$$\models \Gamma \vdash \{u\}\{w\}\{hPre := \text{heap}\}\{v\}(post \land \text{reachableState}' \rightarrow [\ldots] \varphi), \Delta \tag{11}$$

and if for all types $B \subseteq A$ we have

$$\models \text{CorrectMethodContract}(mct, B), \tag{12}$$

then the following holds:

$$\models \Gamma \vdash \{u\}[r = o.m(p'_1, \ldots, p'_n); \ldots [\varphi], \Delta. \tag{13}$$

Proof. Let (10), (11) and (12) hold. Let furthermore $\mathcal{K} = (D, \delta, I, S, \rho)$ be a Kripke structure, $s \in S$, and $\beta$ be a variable assignment. Our goal is to show

$$(\mathcal{K}, s, \beta) \models \Gamma \vdash \{u\}[r = o.m(p'_1, \ldots, p'_n); \ldots [\varphi].$$

If there is $\gamma \in \Gamma$ with $\text{val}_{\mathcal{K}, s, \beta}(\gamma) = \text{ff}$ or if there is $\delta \in \Delta$ with $\text{val}_{\mathcal{K}, s, \beta}(\delta) = \text{tt}$, then this is trivially true. We therefore assume that

$$(\mathcal{K}, s, \beta) \models \bigwedge (\Gamma \cup -\Delta), \tag{13}$$

and aim to show that $(\mathcal{K}, s, \beta) \models \{u\}[r = o.m(p'_1, \ldots, p'_n); \ldots [\varphi].$

Let $s_1 = \text{val}_{\mathcal{K}, s, \beta}(u)(s)$. Then our goal is to show

$$(\mathcal{K}, s_1, \beta) \models [r = o.m(p'_1, \ldots, p'_n); \ldots [\varphi].$$

Let $s_2 = \text{val}_{\mathcal{K}, s_1, \beta}(w)(s_1).$ Because of the definition of $w$, it holds for all $a \in PV \setminus \{this, p_1, \ldots, p_n\}$ that $s_1(a) = s_2(a).$ Since by Def. 4 neither this nor $p_1, \ldots, p_n$ occur in the above formula, Prop. 1 tells us that the interpretation of this formula is the same in $s_1$ and $s_2$. It is therefore sufficient if we show

$$(\mathcal{K}, s_2, \beta) \models [r = o.m(p'_1, \ldots, p'_n); \ldots [\varphi].$$

The definition of $w$ and Prop. 1 ensure that $s_2(this) = \text{val}_{\mathcal{K}, s_2, \beta}(o)$, and that $s_2(p_1) = \text{val}_{\mathcal{K}, s_2, \beta}(p'_1)$, ..., $s_2(p_n) = \text{val}_{\mathcal{K}, s_2, \beta}(p'_n).$ Thus, we can aim to prove the formula below instead of the formula above:

$$(\mathcal{K}, s_2, \beta) \models [r = this.m(p_1, \ldots, p_n); \ldots [\varphi].$$

Since by Def. 4 the program variable res does not occur in the above formula, the Java semantics allows us to instead show

$$(\mathcal{K}, s_2, \beta) \models [\text{res} = this.m(p_1, \ldots, p_n); \ r = \text{res} \ldots [\varphi].$$
Let \( s_3 = val_{\mathcal{K},s_2,\beta}(hPre := \text{heap})(s_2) \). Since by Def. 4 the program variable \( hPre \) does not occur in the above formula, by Prop. 1 it is sufficient if we prove

\[
(\mathcal{K}, s_3, \beta) \models [\text{res} = \text{this.m(p}_1,\ldots,\text{p}_n); \ r = \text{res}; \ \ldots;] \varphi. \quad \text{(thm1-goal)}
\]

We combine (13) with (10) to get

\[
(\mathcal{K}, s, \beta) \models \{u\}\{w\}(\text{pre} \land \text{reachableState}),
\]

which by definition of \( s_2 \) is the same as

\[
(\mathcal{K}, s_2, \beta) \models \text{pre} \land \text{reachableState}. \quad \text{(14)}
\]

Let \( C = \delta(s_2(\text{this})) \). This means that

\[
(\mathcal{K}, s_2, \beta) \models \text{exactInstance}_C(\text{this}). \quad \text{(15)}
\]

Since \( \alpha(\text{this}) = A \), we have \( C \subseteq A \) because of well-typedness. Instantiating (12) with \( C \) and \( s_2 \) yields

\[
(\mathcal{K}, s_2, \beta) \models \text{pre} \land \text{reachableState} \land \text{exactInstance}_C(\text{this}) \rightarrow \{\text{hPre} := \text{heap}\}[\text{res} = \text{this.m(p}_1,\ldots,\text{p}_n);]''(\text{post} \land \text{frame})
\]

where \([\cdot]'\) is \( \langle \cdot \rangle \) if \([\cdot]\) is \( \langle \cdot \rangle \), and where \([\cdot]'\) is either \( \langle \cdot \rangle \) or \( \lfloor \cdot \rfloor \) otherwise. Together with (14) and (15), this implies

\[
(\mathcal{K}, s_2, \beta) \models \{\text{hPre} := \text{heap}\}[\text{res} = \text{this.m(p}_1,\ldots,\text{p}_n);]''(\text{post} \land \text{frame}).
\]

With the definition of \( s_3 \), this becomes

\[
(\mathcal{K}, s_3, \beta) \models [\text{res} = \text{this.m(p}_1,\ldots,\text{p}_n);]''(\text{post} \land \text{frame}). \quad \text{(16)}
\]

If there is no \( s_4 \in \mathcal{S} \) such that \((s_3, s_4) \in \rho(\text{res} = \text{this.m(p}_1,\ldots,\text{p}_n);) \) (i.e., if the method call does not terminate when started in \( s_3 \)), then (16) implies that \([\cdot]'\) must be \( \lfloor \cdot \rfloor \), and thus \([\cdot]\) also must be \( \lfloor \cdot \rfloor \). Then, (thm1-goal) holds trivially, because there is no final state which would have to satisfy \( \varphi \).

We can thus find \( s_4 \in \mathcal{S} \) such that \((s_3, s_4) \in \rho(\text{res} = \text{this.m(p}_1,\ldots,\text{p}_n);) \). As our programs are deterministic, \( s_4 \) is the only such state. Our proof goal (thm1-goal) now becomes

\[
(\mathcal{K}, s_4, \beta) \models [r = \text{res}; \ \ldots;] \varphi. \quad \text{(thm1-goal')}\]

From (16) and the definition of \( s_4 \) we get

\[
(\mathcal{K}, s_4, \beta) \models \text{post} \land \text{frame}. \quad \text{(17)}
\]

Let \( \text{noDeallos} \in Fma_\Sigma \) be as in Def. 7. Prop. 4 tells us that

\[
(\mathcal{K}, s_2, \beta) \models \text{reachableState} \rightarrow \{\text{hPre} := \text{heap}\}[\text{res} = \text{this.m(p}_1,\ldots,\text{p}_n);]''(\text{reachableState}' \land \text{noDeallos}).
\]
Together with (14) and the definition of $s_4$, this turns into
\[
(\mathcal{K}, s_4, \beta) \models \text{reachableState'} \land \text{noDeallocs}.
\] (18)

Let $\mathcal{K}' = (\mathcal{D}, \delta', I', S, \rho)$ be a Kripke structure identical to $\mathcal{K}$, except that $I'(h) = s_4(\text{heap})$, and except that $I'(r') = s_4(\text{res})$. Since by Def. 4 the symbols $h$ and $r'$ do not occur in $\Gamma$ nor in $\Delta$, we get from (13) that $(\mathcal{K}', s, \beta) \models \bigwedge (\Gamma \cup \neg \Delta)$. This and (11) imply
\[
(\mathcal{K}', s, \beta) \models \{u\} \{w\} \{\text{hPre} := \text{heap}\} \{v\} (\text{post} \land \text{reachableState'} \rightarrow \Box \Box \Box \Box \varphi).
\]

As $h$ and $r'$ do not occur in $u$, in $w$ or in $\text{hPre} := \text{heap}$, the above and Prop. 2 imply that
\[
(\mathcal{K}', s_3, \beta) \models \{v\} (\text{post} \land \text{reachableState'} \rightarrow \Box \Box \Box \Box \varphi).
\]

Let $s'_4 = \text{val}_{\mathcal{K}', s_3, \beta}(v)(s_3)$. Then the above implies
\[
(\mathcal{K}', s'_4, \beta) \models \text{post} \land \text{reachableState'} \rightarrow \Box \Box \Box \Box \varphi.
\]

Since $h$ and $r'$ do not occur in the above formula, by Prop. 2 we get that
\[
(\mathcal{K}, s'_4, \beta) \models \text{post} \land \text{reachableState'} \rightarrow \Box \Box \Box \Box \varphi.
\] (19)

Given the definition of $s_4$, the semantics of Java tells us that for all $a \in \mathcal{P}\mathcal{V} \setminus \{\text{heap, res}\}$ we have $s_4(a) = s_4(a)$. Similarly, the definition of $s'_4$ implies that for all $a \in \mathcal{P}\mathcal{V} \setminus \{\text{heap, res}\}$ we have $s_3(a) = s'_4(a)$. Together, we have

for all $a \in \mathcal{P}\mathcal{V} \setminus \{\text{heap, res}\}$ : $s'_4(a) = s_4(a)$. (20)

The definition of $s'_4$ also guarantees that
\[
s'_4(\text{heap}) = \text{val}_{\mathcal{K}', s_3, \beta}(\text{anon(\text{heap, mod}, h)}) \quad (21)
s'_4(\text{r}) = I'(r') = s_4(\text{res}) \quad (22)
s'_4(\text{res}) = I'(r') = s_4(\text{res}) \quad (23)
\]

Using (17) and (18), Lemma 1 tells us that
\[
(\mathcal{K}, s_4, \beta) \models \text{heap} \equiv \text{anon(hPre, \{\text{heap := hPre}\} mod, heap)},
\]

which we can also express as
\[
s_4(\text{heap}) = \text{val}_{\mathcal{K}', s_4, \beta}(\text{anon(hPre, \{\text{heap := hPre}\} mod, heap)}).
\]

Since by Def. 4 the function symbols $h$ and $r'$ do not occur in the above formula, and since $\mathcal{K}'$ is otherwise identical to $\mathcal{K}$, Prop. 2 yields
\[
s_4(\text{heap}) = \text{val}_{\mathcal{K}', s_4, \beta}(\text{anon(hPre, \{\text{heap := hPre}\} mod, heap)}).
\]
As we defined $K'$ such that $I'(h) = s_4(\text{heap})$, this implies

$$s_4(\text{heap}) = \text{val}_{K',s_4,\beta}(\text{anon}(\text{hPre}, \{\text{heap} := \text{hPre}\}) \mod h)).$$

Since $s_3$ and $s_4$ are identical except for $\text{heap}$ and $\text{res}$, and since $\text{res}$ does not occur in $\{\text{heap} := \text{hPre}\} \mod$, Prop. 3 tells us that $\text{val}_{K',s_4,\beta}(\{\text{heap} := \text{hPre}\}) = \text{val}_{K,s_3,\beta}(\{\text{heap} := \text{hPre}\})$. As $\text{heap}$ and $\text{res}$ do not occur in the other arguments of $\text{anon}$, we can transform the statement above into

$$s_4(\text{heap}) = \text{val}_{K',s_3,\beta}(\text{anon}(\text{hPre}, \{\text{heap} := \text{hPre}\}) \mod h)).$$

The definition of $s_3$ implies $s_3(\text{heap}) = s_3(\text{hPre})$. Thus, the update $\text{heap} := \text{hPre}$ has no effect in $s_3$. This allows simplifying the above into

$$s_4(\text{heap}) = \text{val}_{K',s_3,\beta}(\text{anon}(\text{hPre}, \text{mod} \mod, h)).$$

This, together with (21), implies that $s_4(\text{heap}) = s_4'(\text{heap})$. Combining this result with (20) and (23) yields that $s_4$ and $s_4'$ differ at most in $r$. Since by Def. 4 the program variable $r$ does not occur in $\text{post}$, (17) and Prop. 1 imply

$$(K, s_4', \beta) \models \text{post}. \quad (24)$$

As $r$ also does not occur in $\text{reachableState}'$, we get from (18) that

$$(K, s_4', \beta) \models \text{reachableState}'. \quad (24)$$

This, (24) and (19) together imply

$$(K, s_4', \beta) \models [\ldots] \varphi.$$  

By (22) and (23), we know that $s_4'(\text{res}) = s_4'(r)$. Thus, the Java semantics allows us to rewrite the above statement into

$$(K, s_4', \beta) \models [r = \text{res}; \ldots] \varphi.$$  

Finally, as $s_4$ and $s_4'$ differ at most in $r$, Prop. 3 tells us that

$$(K, s_4, \beta) \models [r = \text{res}; \ldots] \varphi,$$

and this is property (thm1-goal') which we aimed to show.  

\[\square\]

2.3 Dependency Contracts

**Theorem 2 (Soundness of useDependencyContract).** Let $\Gamma, \Delta \in 2^{\text{Fma}_{\Sigma}}$, $\text{obs} \in \mathcal{F} \cup \mathcal{P}$, $h^\text{new} = (f_1(f_2(\ldots (f_m(\text{hbase}, \ldots))))), o, p'_1, \ldots, p'_n \in \text{Trm}_{\Sigma}$, $A \in \mathcal{T}$, $\text{depcnt} = (\text{obs}, \text{this}, (p_1, \ldots, p_n), \text{pre}, \text{dep})$, $\text{hPre} \in \mathcal{PV}$, $\text{mod} = \text{allLocs} \setminus \text{dep},$
reachableState, frame, noDeallocs ∈ Fma\_Σ, w ∈ Upd\_Σ, guard, equal ∈ Fma\_Σ all be as in Def. 7 of [2]. If

$$\models \Gamma, \ guard \rightarrow \ equal \Rightarrow \ \Delta$$

and if for all types \(B \subseteq A\) we have

$$\models \ CorrectDependencyContract(\depct, B),$$

then the following holds:

$$\models \ \Gamma \Rightarrow \ \Delta.$$

Proof. Let (25) and (26) hold, and let \(K = (D, \delta, I, S, \rho)\) be a Kripke structure. Our goal is to show \((K, s, \beta) \models \Gamma \Rightarrow \Delta\). We will do a proof by contradiction and assume that this does not hold, or in other words, that \((K, s, \beta) \not\models (\Gamma \cup \neg \Delta)\) holds. This and (25) imply \((K, s, \beta) \not\models (\text{guard} \rightarrow \text{equal})\), which means that \((K, s, \beta) \models \text{guard} \land \neg \text{equal}\). If we insert the definitions of \text{guard} and \text{equal}, and distribute the update \(w\) over the conjuncts of \text{guard}, then this reads as

\[
(K, s, \beta) \models \{w\}\{\text{heap} := h^\text{base}\}(\text{pre} \land \text{reachableState})
\]

\[
(K, s, \beta) \models \{w\}\{\text{hPre} := h^\text{base} \parallel \text{heap} := h^\text{new}\}(\text{frame} \land \text{noDeallocs})
\]

\[
(K, s, \beta) \models \neg (\text{obs}(h^\text{new}, o, p'_1, \ldots, p'_n) \equiv \text{obs}(h^\text{base}, o, p'_1, \ldots, p'_n))
\]

(27)

Let \(s_1 = \text{val}_{K, s, \beta}(w)(s)\). Then the first two statements above become

\[
(K, s_1, \beta) \models \{\text{heap} := h^\text{base}\}(\text{pre} \land \text{reachableState})
\]

\[
(K, s_1, \beta) \models \{\text{hPre} := h^\text{base} \parallel \text{heap} := h^\text{new}\}(\text{frame} \land \text{noDeallocs})
\]

Let \(s^\text{base}_1 = \text{val}_{K, s, \beta}(\text{heap} := h^\text{base})(s_1)\), \(s^\text{new}_1 = \text{val}_{K, s, \beta}(\text{hPre} := h^\text{base} \parallel \text{heap} := h^\text{new})(s_1)\). Then the statements above turn into

\[
(K, s^\text{base}_1, \beta) \models \text{pre} \land \text{reachableState}
\]

\[
(K, s^\text{new}_1, \beta) \models \text{frame} \land \text{noDeallocs}
\]

(28)

(29)

As \(\text{this}, p_1, \ldots, p_n\) do not occur in (27), and as \(s\) and \(s_1\) are otherwise identical, we get by Prop. 1 that

\[
(K, s_1, \beta) \models \neg (\text{obs}(h^\text{new}, o, p'_1, \ldots, p'_n) \equiv \text{obs}(h^\text{base}, o, p'_1, \ldots, p'_n)),
\]

which because of the definition of \(s_1\) implies that

\[
(K, s_1, \beta) \models \neg (\text{obs}(h^\text{new}, \text{this}, p_1, \ldots, p_n) \equiv \text{obs}(h^\text{base}, \text{this}, p_1, \ldots, p_n)).
\]

(30)

Lemma 1 and (29) tell us that

\[
(K, s^\text{new}_1, \beta) \models \text{heap} \doteq \text{anon}\{\text{hPre}, \{\text{heap} := h^\text{Pre}\}\text{mod}, \text{heap}\},
\]

(31)
which because of the definition of $s^\text{new}_1$ is the same as

$$(K, s_1, \beta) \models h^{\text{new}} = \text{anon}(h^{\text{base}}, \{\text{heap} := h^{\text{base}}\} \mod h^{\text{new}}).$$

(31)

Let $C = \delta(s^{\text{base}}_1(\text{this}))$. This means that

$$(K, s^{\text{base}}_1, \beta) \models \text{exactInstance}_C(\text{this}).$$

(32)

Let $K' = (D, \delta', I', S, \rho)$ be a Kripke structure identical to $K$, except that $I'(h) = \text{val}_{K, s_1, \beta}(h^{\text{new}})$. Since $\alpha(\text{this}) = A$, we have $C \subseteq A$. Instantiating (26) with $C$, $K'$ and $s^{\text{base}}_1$ yields

$$(K', s^{\text{base}}_1, \beta) \models \text{pre} \land \text{reachableState} \land \text{exactInstance}_C(\text{this}) \rightarrow \text{obs}(\text{heap}, \text{this}, p_1, \ldots, p_n)$$

$$\equiv \{\text{heap} := \text{anon}(\text{heap}, \mod, h)\} \text{obs}(\text{heap}, \text{this}, p_1, \ldots, p_n).$$

As $h$ does not occur in (28) or (32), we have $\models (K', s^{\text{base}}_1, \beta) \models \text{pre} \land \text{reachableState} \land \text{exactInstance}_C(\text{this})$ by Prop. 2, which we can combine with the statement above to get

$$(K', s^{\text{base}}_1, \beta) \models \text{obs}(\text{heap}, \text{this}, p_1, \ldots, p_n)$$

$$\equiv \{\text{heap} := \text{anon}(\text{heap}, \mod, h)\} \text{obs}(\text{heap}, \text{this}, p_1, \ldots, p_n).$$

Applying the update yields

$$(K', s^{\text{base}}_1, \beta) \models \text{obs}(\text{heap}, \text{this}, p_1, \ldots, p_n)$$

$$\equiv \text{obs}(\text{anon}(\text{heap}, \mod, h), \text{this}, p_1, \ldots, p_n).$$

Because of the definition of $s^{\text{base}}_1$, this is the same as

$$(K', s_1, \beta) \models \text{obs}(h^{\text{base}}, \text{this}, p_1, \ldots, p_n)$$

$$\equiv \text{obs}(\text{anon}(h^{\text{base}}, \{\text{heap} := h^{\text{base}}\} \mod h^{\text{new}}), \text{this}, p_1, \ldots, p_n).$$

By definition of $K'$, we have $I'(h) = \text{val}_{K', s_1, \beta}(h^{\text{new}})$. As $h$ does not occur in $h^{\text{new}}$, and as $K$ and $K'$ are otherwise identical, Prop. 2 guarantees that $\text{val}_{K', s_1, \beta}(h^{\text{new}}) = \text{val}_{K, s_1, \beta}(h^{\text{new}})$. Thus, we have $I'(h) = \text{val}_{K', s_1, \beta}(h^{\text{new}})$, and can thus write the statement above as

$$(K', s_1, \beta) \models \text{obs}(h^{\text{base}}, \text{this}, p_1, \ldots, p_n)$$

$$\equiv \text{obs}(\text{anon}(h^{\text{base}}, \{\text{heap} := h^{\text{base}}\} \mod h^{\text{new}}), \text{this}, p_1, \ldots, p_n).$$

As the function symbol $h$ does not occur in the above formula, ans as $K$ and $K'$ are otherwise identical, Prop. 2 tells us that

$$(K, s_1, \beta) \models \text{obs}(h^{\text{base}}, \text{this}, p_1, \ldots, p_n)$$

$$\equiv \text{obs}(\text{anon}(h^{\text{base}}, \{\text{heap} := h^{\text{base}}\} \mod h^{\text{new}}), \text{this}, p_1, \ldots, p_n).$$

We can combine this with (31) to get

$$(K, s_1, \beta) \models \text{obs}(h^{\text{base}}, \text{this}, p_1, \ldots, p_n) \equiv \text{obs}(h^{\text{new}}, \text{this}, p_1, \ldots, p_n),$$

which contradicts (30). □
References
